What is claimed is:

1. A method for generating hardware configuration data from software constructs, the method comprising:

parsing high-level software programming code, wherein the code is transparent with regard to hardware resources and hardware configuration; and

compiling hardware configuration data based on the high-level software programing code.

2. The method of claim 1 wherein compiling hardware configuration data comprises compiling hardware configuration data to implement blocks.

3. The method of claim 2 wherein implementing blocks comprises representing software variables as a set of wires, wherein the set of wires comprise data wires and a computed wire to denote the variable is valid.

4. The method of claim 1 wherein compiling hardware configuration data comprises compiling hardware configuration data that generates hardware capable of pipelining.

5. The method of claim 1 wherein compiling hardware configuration data comprises compiling hardware configuration data for a device selected from a group consisting of a programmable logic device, a field programmable gate array, and a combination thereof.

6. The method of claim 1 wherein compiling hardware configuration data comprises compiling hardware configuration data that generates hardware capable of speculative execution.

7. The method of claim 1 wherein compiling hardware configuration data comprises compiling hardware configuration data that generates hardware capable of making run-time decisions with regard to parallel execution.

8. The method of claim 1 wherein parsing the high-level programming code comprises parsing code selected from C code, C++ code, JAVA code, LISP code, BASIC code, Pascal code, COBOL code, Fortran code, and a combination thereof.

9. A method for generating hardware that exploits parallelism by making decisions at run-time, the method comprising:

generating a control flow in the hardware, wherein the control flow indicates a status for a block, wherein the status indicates a capability for speculation; and

making run-time decisions regarding execution of the block at least partially based on the control flow.

10. The method of claim 9 wherein making run-time decisions comprises deciding to execute the block.

11. The method of claim 9 wherein making run-time decisions comprises deciding not to execute the block.

12. The method of claim 9 wherein making run-time decisions comprises deciding to execute the block speculatively.

13. The method of claim 9 wherein making run-time decisions comprises deciding to execute the block speculatively and in parallel.

14. The method of claim 13 wherein there is no data dependency between blocks being executed speculatively and in parallel.

15. The method of claim 13 further comprising sharing a common variable between blocks being executed speculatively and in parallel, wherein there is no data dependency between the blocks being executed speculatively and in parallel.

16. The method of claim 12 wherein the block is not comprised of any mutable operations.

17. The method of claim 12 wherein the block is comprised of a mutable operation, and wherein the mutable operation overwrites hardware states that are not needed.

18. The method of claim 9 wherein generating the control flow comprises using a software-to-hardware compiler to generate the control flow.

19. The method of claim 9 further comprising implementing a software program in the hardware.

20. A method for optimizing hardware that is generated by a software-to-hardware compiler during compilation, the method comprising:

locating at least one expression in a software program using the software-to-hardware compiler, wherein the expressions are used more than once in the program;

using a single set of hardware resources to implement the expression; and

selecting at run-time instances that will have access to the single set of hardware resources.

21. The method of claim 15 further comprising using a single set of hardware resources to implement a function of the software program.

22. The method of claim 15 wherein the optimizing takes place at a later stage of the compilation.

23. The method of claim 15 wherein the optimizing is transparent to a user.

24. A method for mapping software constructs of a program into hardware constructs, the method comprising:

mapping software constructs into a block of logic operations using a software-to-hardware compiler;

coupling an input environment to the block that carries information into the block; and

coupling an output environment to the block that carries information out of the block.

25. The method of claim 24 wherein the input environment comprises wires that implement elements selected from a group consisting of control flow, variables, arrays, pointers, expressions, a reset signal, and a combination thereof.

26. The method of claim 24 wherein the input environment comprises wires that implement elements selected from a group consisting of control flow, variables, arrays, pointers, expressions, a done signal, and a combination thereof.

27. A method for mapping software constructs into hardware constructs, the method comprising mapping a variable into a set of wires, wherein one of the wires indicates whether the variable has been computed and the remainder of the wires indicate a value of the variable.

28. A programmable logic resource that exploits parallelism by making decisions at run-time, wherein the programmable logic resource is configured to:

generate a control flow in the hardware, wherein the control flow indicates a status for an operation; and

make run-time decisions regarding execution of the operation at least partially based on the control flow.

29. The programmable logic resource of claim 28 that is further configured to decide to execute the operation.

30. The programmable logic resource of claim 28 that is further configured to decide not to execute the operation.

31. The programmable logic resource of claim 28 that is further configured to decide to execute the operation speculatively.

32. The programmable logic resource of claim 28 that is configured by a software-to-hardware compiler to generate the control flow.

33. The programmable logic resource of claim 28 that is further configured to implement a software program.

34. A programmable logic resource that is made of hardware constructs mapped from software constructs of a program into hardware constructs, the programmable logic resource comprising:

blocks of logic operations that are mapped from software constructs using a software-to-hardware compiler;

an input environment that is coupled to the block that carries information into the block; and

an output environment that is coupled to the block that carries information out of the block.

35. The programmable logic resource of claim 34 wherein the input environment comprises wires that implement elements selected from a group consisting of control flow, variables, arrays, pointers, expressions, a reset signal, and a combination thereof.

36. The programmable logic resource of claim 34 wherein the output environment comprises wires that implement elements selected from a group consisting

of control flow, variables, arrays, pointers, expressions, a done signal, and a combination thereof.

37. A hardware construct that is implemented in programmable logic that is a mapping of a software constructs into hardware constructs, the hardware construct comprising a set of wires that is a mapping of a variable, wherein one of the wires indicates whether the variable has been computed and the remainder of the wires indicate a value of the variable.